Plasimo User Guide

The Plasimo Team

September 23, 2015

Contents

Chapter 1

Introduction

This is the Plasimo [1] user guide.

Plasimo is available for Unix and Windows users. For access please fill in the access request form on the Plasimo web page `https://plasimo.phys.tue.nl/form_access_request.html` or contact the Plasimo team `info@plasimo.phys.tue.nl`.

If Plasimo is suitable for your particular application, you need to sign the software license. Plasimo team will provide you with information about the terms and conditions. After obtaining a license you will be able to download the software and you will have access to all relevant documentation on this website.

Chapter 2

Downloading, Installing and Running Plasimo

Plasimo is available in two flavors: stable, released versions, and an experimental developer version, where all the changes are done for the next release.

Before you get the Plasimo sources, you should take a decision which branch you want to use: "Stable" or "Developer" (also called HEAD). "Developer" is recommended for people who develop code for Plasimo or that need the latest features. The downside is that experimental changes are allowed to go in and carry the risk of breaking user input files, producing wrong results or crashing the user interface. Stable is normally more suitable for people that are mainly users of Plasimo, as care is taken to not break stuff for releases and users that need to update their input files are advised how to do so. The file `NEWS` contains the documentation on what has changed since the last release. Note that it's possible to have more than one copy of the Plasimo source tree on your machine, so you can work with several versions if needed.

2.1 Downloading Plasimo

Assumes that you already have the rights for downloading Plasimo and you have decided which version of Plasimo you want to use. The Plasimo packages can be downloaded from the Plasimo's web page `http://plasimo.phys.tue.nl/generated-docs/index.html`

Depending on your operating system proceed to the relevant section below.

For Microsoft Windows users:

Download the compressed zip file for Windows 2000+ binary distribution of the chosen version of Plasimo. Extract the files on your hard drive. We recommend to extract the files in your home directory. A typical installation directory may be `C:\\Users\jan\`. The zip archive puts all files in a subdirectory with the name `Plasimo`, so you end up with a directory `C:\Users\jan\Plasimo`. This directory will be called the *Plasimo root directory* hereafter.

Note that you do not need to have administrator rights to install the modeling package, as long as you have write access to the installation directory. In particular, installing Plasimo does not require changes to the Windows Registry.

For Unix/Linux users: openSUSE

You can use the default Installer that will guide you in the installation procedure or you can save the rpm file and execute the command below after replacing the 'XXX' with the correct rpm file name:

```
zypper install plasimo-XXX-XXX.rpm
```

In both cases root privileges are needed to install Plasimo. The installation directory is `/opt`.

For Unix/Linux users with access to the source code:

The following steps concern users who have CVS access to the code of Plasimo, as provided by Plasimo team.

Before to proceed with the installation and configuration of Plasimo, please be sure that you have checked carefully the system requirements and that you have installed all components mentioned in appendix A.

The following steps demonstrate how to check out and build an optimized version of Plasimo that includes graphical user-interface (GUI) support.

Firstly, open a shell (console) and change to the directory where you want to create the Plasimo subdirectory. Next type the following sequence of commands (replace `NAME` with your login name on the server). Please note that all commands and options under Unix are case-sensitive.

- `export CVS_RSH=ssh`
- `cvs -d :ext:NAME@plasimo.phys.tue.nl:/usr/local/src/cvs checkout gum`
- `cd gum`
- `./makeconf.sh`
- `mkdir linux-opt`
- `cd linux-opt`
- `../configure --prefix=~/plasimo-opt`
- `make`

Assuming everything went fine, you are now ready to run Plasimo. In case any of the previous steps fails, please check if you have all prerequisites installed, or consult the appendix sections A and B for a possible solution.

Additional information and in-depth discussion of these download and installation processes are provided in appendix B. After building Plasimo, you may wish to fine-tune the behavior of Plasimo, according to your personal preferences. This is also discussed in section B.

2.2 Starting the Application

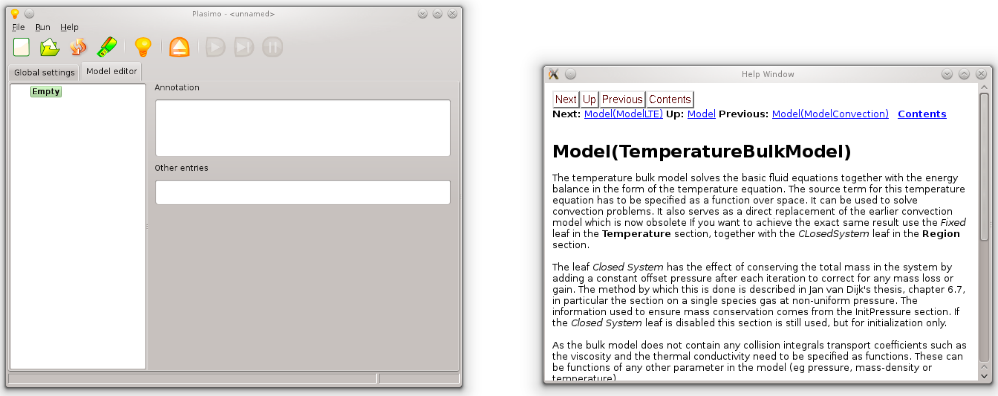
Execution on Windows:

In order to facilitate starting the executables under an MS Windows environment, a set of batch files is shipped with the distribution, these can be found in the Plasimo root directory.

You can start the version of Plasimo with the graphical user-interface by simply executing the script `Plasimo.bat`, which resides in the PLASIMO root directory, for example with Windows Explorer. Together with two console windows the PLASIMO main window appears. The second console window shows the log of PLASIMO. After starting the application with an 'empty' model, you should see a window like that in figure 2.1(a).

Execution on Linux/Unix:

As part of the build process, some information about the location of various files is recorded and stored in two scripts, `set-plasimo-local` and `set-plasimo`. The first is used when one wishes to run the version of Plasimo at the location where it was build, the latter when one wishes to use an installed version of Plasimo.



(a) Plasimo graphical user interface

(b) Help window

Figure 2.1: Two screen-shot of Plasimo's graphical user interface (GUI). The screen-shot of the main window (left figure) has been taken just after starting the application, when no model file has been loaded yet. The elements of the main window (left figure) are a menu bar, a tool button bar and set of tab windows. In the figure a tab window called 'model editor' is active. Context-sensitive help text is displayed in a second window (right), a typical example is shown in the figure on the right.

- Installed version of Plasimo:
  Execute the following commands from your home directory:

  1. `. /opt/plasimo/bin/set-plasimo`
     or `source /opt/plasimo/bin/set-plasimo`
  2. `/opt/plasimo/bin/wxplasimo`

  The fist command sets the current working directory as `/home/jan`. Before continuing with loading a model, you have to prepare a directory where Plasimo can write the output data. You have to create a directory `data` in your home directory:

  ```
  mkdir data
  ```

- Built version of Plasimo:
  Let us assume that we wish to run the version of Plasimo at the build location (`gum/linux-opt/`). This can be done by entering the following commands in the build directory:

  1. `. set-plasimo-local`
     or `source set-plasimo-local`
  2. `./app/wx/wxplasimo`

After starting the application with an 'empty' model, you should see a window similar to the one shown in figure 2.1(a).

2.3 Installing and running a model

In the application main window you see a menu. In order to allow quick access to the menu items, most commands are also available through the *toolbar*. Initially, the buttons **New**, **Open**, **Reload**, **Save**, **Help** and **Install** are active. While the icons are easily recognized, in case of doubt you may move the mouse over a button to see what it does: after a short time a 'tooltip'

will be displayed. The remainder of the main window displays the properties of the active model. We will get back to that later.

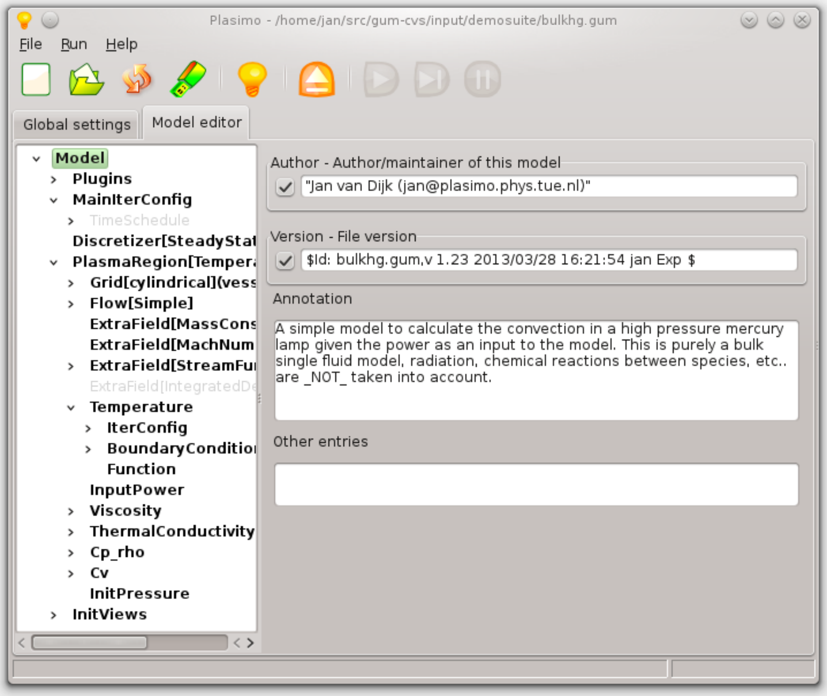


Figure 2.2: A screen-shot of Plasimo's graphical user interface (GUI). The screen-shot has been taken just after loading an existing model, named bulkhg.gum.

1. **New** button gives you the possibility to create a new model by selecting a model type from the pop-up menu.

2. **Open** pops up a file dialog which allows you to load a model by selecting an existing input file. A number of existing models are shipped with the Plasimo distribution. All the Plasimo input files reside in the directory `input/`. The location of this directory depends on the specific Plasimo version you are running:

   - Windows zip: the installation directory;
   - Linux package: in directory `/opt/plasimo-XXX-XXX` (or appropriate softlink `/opt/plasimo`);
   - Linux compiled from source: root source code directory.

   The files in the `input/` directory are separated in sub-folders depending on the sub-modules of Plasimo. Some of the existing example input files are described in more detail in the chapter 4.

3. Please load the example file `input/demosuite/bulkhg.gum`. If there were no errors, the main window should now look like 2.2. In the tree window on the left you see a tree representation of the `bulkhg.gum` model. The window on the right shows the contents of the active tree item. These editors will be discussed in detail in chapter 3, tutorial sessions are provided in chapter 4.

4. The **Reload** button allows you to reload an input file. Plasimo uses simply text files that can be edited with any of your favorite editors. In case you made changes in the this text input file, this provides a quick way to reload the modified model.

5. The **Help** button activates or deactivates the visibility of a help window. By default it is activated. It shows you a detailed description and requirements of the active tree item.

6. After a model has been loaded or modified to the user's needs, it must be prepared for running. Press the **Install** button to do this. This will do all kinds of necessary one-time installations, and perform various checks on the sanity of the model file. Installing a model might therefore take some time, typically up to a few seconds. If anything goes wrong during installation, a message box will be popped up, telling the user what went wrong. After the problems have been solved, one can try to install the model once more.

7. When the model is successfully installed, additional buttons are activated. These are:

   - **Run**. Run the successfully installed model.
   - **Proceed (one step)**. All models are assumed to run in steps. A step can be either a time step, or an iteration. Selecting this item causes a (paused) model to do one such step. This can be useful if you use the application interactively, for example to find out the reasons for divergence of otherwise unexpected behavior.
   - **Stop**. Pauses a running model; this can be useful to inspect intermediate results.

For more information how to modify an existing model and how to manipulate the input data, please follow the steps in 4. Information about all the functionalities if the graphical user interface of Plasimo, please read chapter 3.

Chapter 3

The Graphical User Interface

This chapter discusses the usage of the graphical user interface (GUI), its functionalities such as the various model editors, settings and model specific GUI extensions.

3.1 The Main Window

After starting the application you should see a window like that in figure 2.1(a). This *application main window* contains a number of elements, from top to bottom we have:

- A menu bar, containing the menus **File**, **Run**, **Help**
- A tool bar, discussed in section 2.3
- A set of *tab windows*, named **Global settings** and **Model editor**
- A status bar, displaying the current status of the model **Installed**, **Paused**, **Running**, etc.

As the menu/tool bars were previously discussed in chapter 2.3, the following text provides detailed information about the tab windows **Global settings**, **Model editor** and the additional **Model Data** window, that appears after installing a model, as well as some model specific extensions.

3.2 The Global Settings Window

In the graphical user interface the contents of the global configuration file are visualized in the tab window **Global Settings**. It is important to notice that some of the settings in the configuration file take effect only after the application is closed and restarted, while others take effect while it is still running. Among other things this can be used to fine-tune the behavior of Plasimo (changing the appearance of plot data and the like).

In the interface the active section is explained in the HTML help window. For an overview of all available sections, we refer to the Plasimo user reference guide.

Most people can use plasimo without ever needing to use this window. Therefore, we shall continue this text with a discussion of the model editors and how you can edit a model using the GUI.

3.3 The Model Editor Window

Initially, the `Model editor` window has been activated. It is one of the tab windows previously mentioned and allows the user to create new model or edit existing one. Only one Plasimo model can be active, multiple documents are not supported.

In general, the structure of the **Model editor** window resembles the interface of most file managers, showing the directory (section) tree on the left (**Tree editor**), while the window on the right shows the contents of the active section (**Leaf editor**).

3.3.1 The Tree Editor

The tree view shows the *nodes* or *sections* which are at the basis of the hierarchic structure of Plasimo input files. Please note that the structure is *recursive*, that is, sections may contain (sub)sections themselves. Each section contains a group of conceptually related subsections and variables.

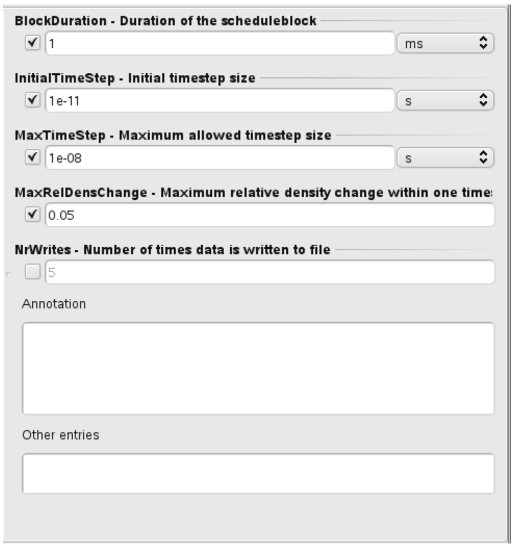
A section can be activated by clicking the left mouse button on the section entry in the tree. The variables which belong to the active subsection are shown in the 'section window' at the right hand side. Furthermore, in the **Help** window a detailed description of this section is displayed.

If you want to add a subsection, click the right mouse button on the section to which you want to add it. This will pop-up a context sensitive menu which allows you to make an appropriate selection. This way it is guaranteed that you add subsections only at locations where the model will actually look for them. By clicking on the check box with the RMB, a pop-up menu appears which offers the following choices:

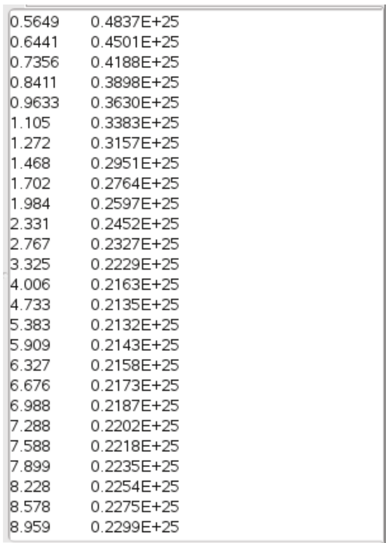
- **Cut** Removes the active node and the contents is copy to the clipboard.
- **Copy** Copy the active node
- **Paste** Paste the copied/cut node
- **Paste into section** Paste a copied node into the active section
- **Disable** this disables the entry if it is enabled. Disabled sections can be recognized by the light-up color
- **Enable** this enables the entry if it is disabled. Pressing the right mouse button over a disabled section pops up a menu which allows to re-enable the section.
- **Undo all changes** This option will undo the results of editing the active node. Node that after switching to another node, editing can no longer be undone.
- **Add...** Add a relevant context dependent section. It is possible to have multiple choices. After one is selected a small addition window appears with the following options:
  - **Create default section** This will create a section with the default specifications. Note, that the relevant subsections will not be created recursively.
  - **Copy file into a section** This option allows you to copy data structures from external files. It works like this: the model will copy the data from an external file and paste it in your input model file.
  - **Include file as a section** This option also allows the usage of external data files. But it works slightly different: the model will only use the external data without writing the whole data it in your model file. Instead, you will see in your input file in the relevant section a line like this: `Include filename.in SectionName`. This allows to have only one definition of an argon mixture, say, on disk, which can be shared (included) by all models that describe some argon plasma.

3.3.2 The Leaf Editors

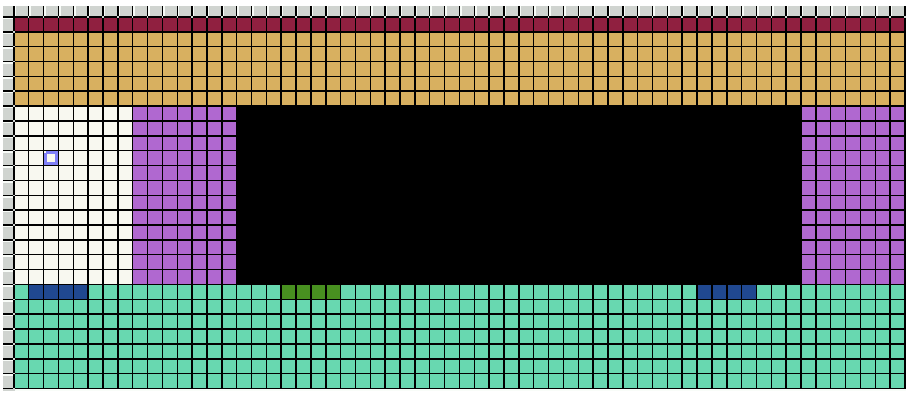
At the right-hand side of the model editor window, an appropriate editor for the data items of the active node are shown. Depending on the node contents, you will see a **Dialog editor**, a **Grid editor** or a **Text editor** window. In figure 2.2 a dialog editor is visible. The various editors are displayed in figure 3.1 and will be discussed next.



(a) Dialog Editor



(b) Text Editor



(c) Grid Editor

Figure 3.1: The various leaf editor windows. Figure 3.1(a) shows the dialog editor. The example shows the time settings of a time-dependent model edited using the dialog editor. Figure 3.1(b) shows a text editor, as an example is a lookup table. Figure 3.1(c) shows the grid editor in action (here a PDP geometry is being edited).

The Dialog Editor

The node dialog editor window (figure 3.1(a)) is the one you will most commonly encounter. It contains one *leaf block* for each data item type. The name of the data item and a short description are displayed above the box. Note that a box may be empty. Clicking on the name of the data item with the right mouse button (RMB) pops up a menu which allows you to add a new leaf of the relevant type. The default values will be used.

Each entry has a check box on the left, followed by one or more editor or choice fields. By clicking on the check box with the left mouse button (LMB), the entry can be disabled or enabled. Disabled entries are not taken into account when the model description is interpreted by Plasimo. They appear in grey and cannot be edited.

In addition, two small text editor windows are available. The 'Annotation' widget can be used to provide a description of the active section. If one exists, it will be shown. The window titled 'Other entries' can be used to add entries in text raw form. Plasimo also dumps entries it cannot

handle here when creating a dialog editor, so it should be empty after creating a fresh dialog editor.

The Grid Editor

The grid editor (figure 3.1(c)) facilitates editing of grids ("matrices") consisting of elements which can be represented as strings. Sometimes the matrix is to big to be visualized in the grid editor. You can zoom out/zoom in the matrix by pressing the control key and scrolling the mouse wheel. The matrix consists on cells with different colors as to each color is assigned different material type.

There are two editing modes, the *Grid layout mode* and *Grid cell mode* mode. By default, the cell mode is active. One can switch between these modes via the context menu which is activated by pressing the right mouse button on the grid region.

The *Grid cell mode* deals with the properties of the cells, such as assignment of new material type. Clicking or selecting cells with the left mouse button (LMB) will assign the active color to them (by default this is the color black, which always corresponds to the material with the lowest alphabetic key value). If the control key is down, instead the material under the LMB is picked up as active color. New material color-index pairs can be created via the menu which is activated by clicking the right mouse button and selecting **Select material**. A list of previously defined materials will pops up for selection of material type.

In *Grid layout mode* only entire rows or columns can be selected. Row and column selection can be activated by clicking on the grey label bar at the left and top side of the grid, respectively. The presently selected row(s) or column(s) can be cut and copied via the menu which is activated by clicking the right mouse button. Then, the cut or copied row(s)/column(s) can be pasted before the active row/column.

Finally, after editing the geometry matrix, you may want to keep or discard the changes. The changes will be automatically saved after switching to an other node. If you do not want to save the changes you made, you can retrieve the original matrix by pressing the RMB on the **GeometryMatrix** in the tree editor and selecting **Undo all changes** from the menu. Note, that after switching to another node, editing can no longer be undone.

The Text Editor

The text editor window allows the user to manipulate the data entries in raw format. In some cases the creation of a text editor window is requested explicitly, because the nature of the data does not allow convenient editing inside a node dialog or grid editor dialog. Perhaps an appropriate editor will be provided later. An example is a look up table in a two-column format (see figure 3.1(b)).

The text editor is also used as fallback option if the creation of a node dialog or grid editor window fails for some reason. If the latter happens, the user will be informed and is kindly requested to send a bug report to the Plasimo maintainers.

There are a few things to think of when using the text editor window. Firstly, disabled leafs occur in raw text format with an underscore (_) prepended. Secondly, make sure to quote items inside this entry which consist of multiple words. As an example, an entry which specifies a name should be written as `Name "Jan van Dijk"` instead of `Name Jan van Dijk`. In the latter case, you will perhaps get an error when Plasimo interprets this entry, but more likely the part `van Dijk` will be silently ignored. The text control correctly handles special characters, so newlines, tabs and backslashes can be safely used.

3.4 The Data Window

After installing the model, an additional tab windows has appeared, named **Model data**. You will see that the available data in this window have been organized as a tree, displayed on the left. The tree consists of a model-dependent data, which may be grouped into sections of related

variables. As a minimum, a section called **General** has been created for every Plasimo model. Not surprisingly, its contents provide some general information about the model, such as the status of the calculation (you may think of data like elapsed time and the number of iterations). This can be activated by double clicking on the item **Common data** inside section **General**.

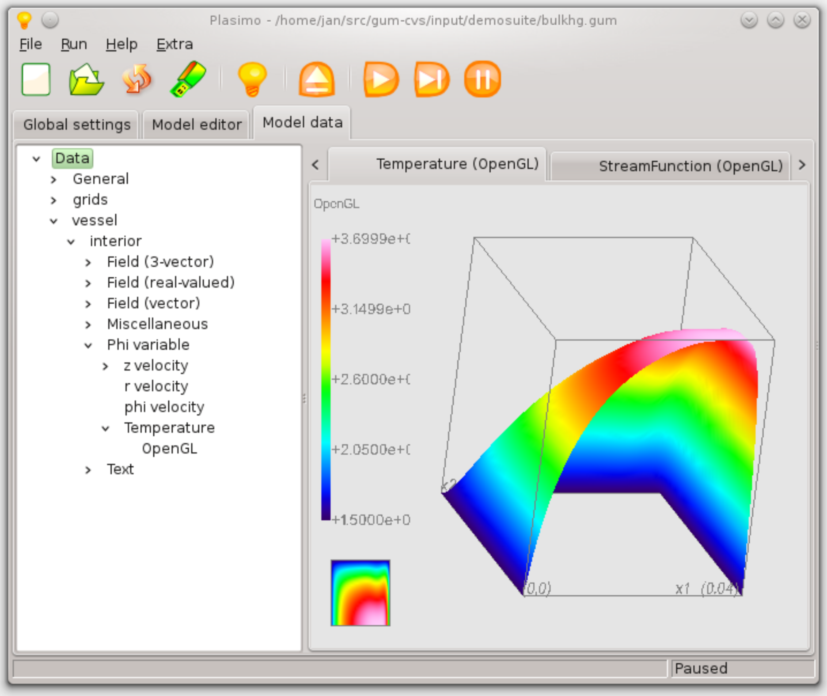


Figure 3.2: A screen-shot of Plasimo's graphical user interface (GUI). The screen-shot has been made during a run of the model bulkhg.gum and shows an OpenGL plot of the temperature of the plasma region.

For most of the Plasimo models, of particular interest is the section **Interior - Phi variable**, which contains some important primary plasma parameters like the bulk velocity, pressure, densities.

For each data item a "viewer" can be created. Creating a viewer can be easily achieved by moving the mouse over the data item 'Temperature', say, and double click on it will open the default OpenGL viewer and you get the result shown in figure 3.2. The viewers are visible at the right hand side of the data window, and are displayed as a series of tab windows. Please take a moment to see what data are there.

Additional viewers are available and you can choose one by clicking the right mouse button in the selected item. This will pop up a menu which offers the option 'Add view'. Some additional functionalities of the various viewers are discussed in the following section.

3.4.1 Additional functionalities of the viewers

Some additional functionalities of the various viewers are available. The next text discusses the different viewers available in Plasimo. Depending on the data item different viewers are available. Plasimo will offer you the appropriate suggestions for the relevant data item.

OpenGL

The OpenGL viewer is the default viewer in Plasimo for plotting 2D and 3D plots. The plot can be rotated by pressing the left mouse button on the plot and drag the plot until the final orientation is reached. The same effect can be obtained using the arrow keys. If you keep the shift key pressed during such operations, the plot will be translated, rather than rotated. The plot window can also be controlled with the following keys:

f – Switch between solid and wire frame plot.
r – Reset the configuration parameters to their default values.
L – Toggle lighting.
s – Toggle smoothing.
o – Include the origin in the plot.
p – Print the plot window to a postscript file.
d – switch between the directions
c – switch between the components
C – Toggle enabling (all) clipping planes
<, > – Decrease/increase the number of planes that is drawn (3D only)
X,Y,Z – This is available only for 3D plots. Draw planes with constant first, second or third coordinate.
x , y , z – Activate the x, y or z coordinate. All subsequent coordinate-specific commands will affect that coordinate. The commands below are all coordinate-specific:
D – Open a dialog window that allows you to configure the way the active coordinate (x, y or z) is handled inside the plot. You can control the coordinate range and the usage of a log scale. A help window is available inside the dialog window.
a – Increase the position of the lower clipping plane.
A – Decrease the position of the lower clipping plane.
b – Increase the position of the upper clipping plane.
B – Decrease the position of the upper clipping plane.
l – Toggle between linear and logarithmic axes.

MathPlot WireFrame

This viewer is suitable for plotting data in a X-Y format. This is the default viewer in Plasimo for plotting this type of data. Pressing the right mouse button, the pop up menu will give you a list of all options available for this plot.

UC WireFrame

This viewer is an other option for plotting data in a X-Y format. Pressing the right mouse will give you the useful option to export the data of the plot as ASCII.

PlotMtv

Choosing this option, Plasimo will prepare the data to be plot using PlotMtv program. Of course, this requires that you have installed the program PlotMtv. Then the plot will be visualized in external PlotMtv window. The viewer is suitable especially for vector plots.

Grid

This viewer shows the data for one field variable as a spreadsheet. The cells in the spreadsheet are placed to resemble the positions of the corresponding data field variable in the master grid. This means that for a cell showing the value of a field defined at the nodal points (e.g. the species densities), the north, south, west and east cells are empty, because in the master grid they represent the positions of the flux field variables. See the grid document for more information.

3.4.2 Initial Viewers

Sometimes, in the working process you may need to install the program multiple times and to monitor some data item at different conditions for example. After each Install of the program you have to open the relevant viewer to monitor the data item in run time. This could be annoying sometimes, so that Plasimo offers the option to avoid this by setting initially the viewers of the variables you want to monitor. As a result, when you install the model, the viewers will be initially open with the set of specifications.

Assume that you want a viewer for "Temperature" data item, as it is shown in figure 3.2, to be initially open always when you install the model. This can be done by adding the optional section **InitViews** in the model tree. Press with right mouse button on the **Model** and select **Add InitViews** from the pop up menu. Next, you have to add the relevant subsection: press the rmb on **InitViews** and add default section **ViewList**. On the leaf dialog editor on the right hand side of the window press again the rmb on the Viewer leaf to add one. Adding a new viewer leaf will displays 4 empty boxes. In the first one you have to specify the data location, for example *vessel/interior/Phi variable* In the second box you have to specify the name of the data item *Temperature*. In the third you can specify the viewer type. If it is empty, the default viewer will be open. In the last box optionally you can specify some attributes, such as min and max values of the x, y or z coordinate and logarithmic scale. All the attributes must be separated by comma, for instance: xmin=0, xmax=5, ymin=3, ymax=20, zmin=1e12, zmax=1e20, You can add viewers leafs as many as you can/want. Note that the editor is case sensitive!

3.5 Model-specific GUI Extensions

In section 2.3 it was mentioned that in addition to the functionality which is shared by all GUM simulation types, a number of application-specific extensions may be present. There are two such extensions available:

- Grid resizing: the grid can be re-dimensioned on the fly;
- Adjustment of Φ-variables: Plasimo models allow the inspection and adjustment of the iterative control parameters of the convection-diffusion (Φ) variables while a simulation is in progress.

If the model supports these functionalities, after installing it, you will see an additional file menu called **Extra**. This menu manages the specified functionalities.

Grid Resizing

Grid resizing can be seen as a *poor man's* implementation of the *multi-grid algorithm*. Essentially, in such methods the number of grid cells is not kept constant. Rather, the results of rapidly converging but inaccurate calculations on coarse meshes are combined with accurate, yet slowly converging calculations on finer meshes. The result is an net reduction of the time to convergence. Typically a number of grid changes is made in both directions. A full multi-grid algorithm has not been implemented in Plasimo. However, wxPlasimo allows its users to manually adjust the mesh size while a simulation is in progress. After a grid resize all grid-dependent variables are interpolated to the new grid and the iterative process resumes.

Please note that this grid resizing can also be configured to be done automatically at the input file level, see section MainIterConfig of chapter 'Input File Reference' of the User Reference Guide for details.

Φ-Variable Adjustment

This allows the adjustment of the iterative control variables of the Φ-variables which are part of the model. More specifically, the user is allowed to set the actual value of the Φ-variables' *under-relaxation factor* (urf). The urf is the fractional amount of the field correction which is determined in an iteration which is actually applied to the old value of the field. Hence, when urf is unity the correction will be completely applied, while for smaller values only a partial adjustment is done. There is a trade-off between speed and robustness here: while smaller under-relaxation factors tend to stabilize the simulation, the time-to-convergence may be considerably longer.

If you select "Φ-Variable Adjustment" from the **Extra** menu in run time, a dialog appears which allows you to select a Φ-variable, if any are present. If one is selected, a second box appears which allows to change the value of the under-relaxation factor, the actual value is proposed as default. Note that in Plasimo, depending on the setting for the urf-adjustment, the relaxation factors are continuously adjusted. Also a urf you set in this dialog is subject to later change.

Secondly, the option 'Freeze' can be activated. The effect is the same as that of a zero under-relaxation factor: no update is done whatsoever. Prefer to enable Freeze above setting a zero urf: in the former no calculation will be done for the variable, considerably saving computing time. Additionally, the discretization of some of the variables relies on the assumption of a non-zero urf.

Of course freezing the variables cannot lead to physically realistic results, this option should therefore not be used in production work. Rather, it is meant to facilitate the analysis of the physical inter-dependencies between the model equations; in particular it can be useful in the study of convergence problems in the model development stage.

Chapter 4

Plasimo example models

4.1 Introduction

This chapter explains some of the existing demo models that were shipped with the Plasimo distribution. The examples give also information how you can edit an existing model and create a new one using the graphical user interface.

4.2 Drift-Diffusion Model

In this section an example input file is used to show how the drift-diffusion module is used.

Example input file: `demo.md2d`

Location: `input/md2d/`

This input file contains a model of a low pressure DC discharge with a simple geometry: two parallel electrodes at a distance of 50 cm, enclosed in a glass tube with a diameter of 5 cm. To keep the model as simple as possible we will start using Cartesian coordinates. The gas is He and only electron impact ionization is considered. The gas pressure is constant at 1 Torr, and a constant voltage of -400 V is applied. You can find the input file `demo.md2d` in directory `input/md2d/`.

After installing the model, an additional tab "Model Data" appears in the main window. On the left hand side the in the herarchial tree the following sections are present:

- `Discharge Region`:
  This section contains all the calculated variables; the section is subdivided into:
  - Electron energy - this subsection contains the data from the energy equation, such as the mean electron energy, mean energy source, mean energy flux density.
  - Reactions - contains the rates of the specified reactions.
  - Species - contains the calculated variables for each species, such as density, mobility, source, flux.

  An OpenGL plot of a variable can be activated by double-clicking a variable in the tree panel on the left.

- `EM Region`:
  In this section you can see the EM data, such as potential distribution, electric field, surface and volume charge densities.
  An OpenGL plot of a variable can be activated by double-clicking a variable in the tree panel on the left.

- `General`:
  Naturally contains the general, common data, such as the status of the simulation. You can activate the `Model Status` by double-clicking on `Common data`.

- `Geometry`:
  Here you can see the geometry - the configuration and the control volume grids.

Output files

All calculated variables are not only shown in the GUI but are also stored on disk. The data path as well as the number of writes of the output data can be specified in the input. (see `Schedule`) By default, the program store the output files in the main plasimo folder if no data path is specified. The output files:

`history.out`: gives the calculated variables as a function of time. The averaged quantities are written with a frequency specified by the user.

`info.out`: the averaged values, written only once at the end of the simulation.
`info.txt`: the same data as in `info.out`, but written with the user-specified frequency.

`reaction_analysis.out` and `reaction_analysis.txt`: detailed reaction analysis: production and destruction contribution of each reaction for each species. The files are written like `info.out` and `info.txt`.

The following files represent the spacial distribution of the quantities. The data are written as many times as specified in the input file.

`n00.txt`	electron energy density [$J\ m^{-3}$]
`phi00.txt`	electron energy flux density [$W\ m^{-2}$]
`S00.txt`	electron energy source [$W\ m^{-3}$]
`D00.txt`	electron energy diffusion coefficient [$W\ m^{2}$]
`mu00.txt`	electron energy mobility coefficient [$J\ m^{2}\ V^{-1}\ s^{-1}$]
`Relas00.txt`	rate of electron energy loss from elastic collisions [$W\ m^{-3}$]
`epsilon.txt`	mean electron energy [J]
`n01.txt`	density for species 1 [m^{-3}]
`S01.txt`	source for species 1 [$m^{-3}\ s^{-1}$]
`D01.txt`	diffusion for species 1 [$m^{2}\ s^{-1}$]
`mu01.txt`	mobility for species 1 [$m^{2}\ V^{-1}\ s^{-1}$]
`phi01.txt`	flux for species 1 [$m^{-2}\ s^{-1}$]
`R00.txt`	reaction rate for reaction 1 [$m^{-3}\ s^{-1}$]
`K00.txt`	reaction rate coefficient for reaction 1 [$m^{3}\ s^{-1}$]
`Pp01.txt`	power dissipation for species 1 [$W\ m^{-3}$]
`P.txt`	dissipated power density [$W\ m^{-3}$]
`J.txt`	current density [$C\ s^{-1}\ m^{-3}$]
`V.txt`	potential [V]
`E.txt`	electric field [$V\ m^{-1}$]
`Er.txt`	reduced electric field E/p [$V\ m^{-1}\ Pa^{-1}$]
`E_N.txt`	reduced electric field E/N [$V\ m^{2}$]
`rho.txt`	volume charge density [$C\ m^{-3}$]
`sigma.txt`	surface charge density [$C\ m^{-2}$]

The numbering of the species is the same as the order of the input file. i.e. in this case `n01.txt` contains the density distribution of the electrons and `n02.txt` the density distribution of the second defined particle, in this case He^{+}. The same numbering applies for the other variables.

Results and discussion

When the model is loaded, installed, and run, it will be possible to identify several regions in the discharge:

- Cathode fall region (Cathode dark space):
  Most of the potential drop between the electrodes occurs in the cathode dark space. This is a region with a strong electric field due to the positive space charge in front of the cathode. The strong field accelerates ions toward the cathode and electrons toward the negative glow region. The electrons ejected from the cathode are accelerated and gain energies up to the cathode fall.

  Select the `EM region` node for the `Potential`, `Ex`, and `Volume charge Density`, and the `Discharge Region` for the `Flux x` of the ions and electrons, and the mean electron energy.

- Negative glow:
  In the negative glow an intense electron impact ionization occurs. The electrons lose their energy and the E-field drops to almost zero. Select the `Discharge Region` node for the reaction rate and the power dissipation of the electrons.

- Faraday dark space:
  As the electrons have dissipated their energy, excitation and ionization will become less and less frequent, because electrons do not gain energy in the weak field. In this region the longitudinal field gradually increases to the E-field in the positive column.

- positive column:
  In this region the potential gradient is practically constant, and electron impact ionization occurs throughout this region.

Variation of the conditions

Changing of the input parameters can be done in `Model editor` tab. Optionally, you can change the input file using you favorite text editor. After every change it is necessary to install the model again, before it is run. Below are few examples of changes using the graphical user interface.

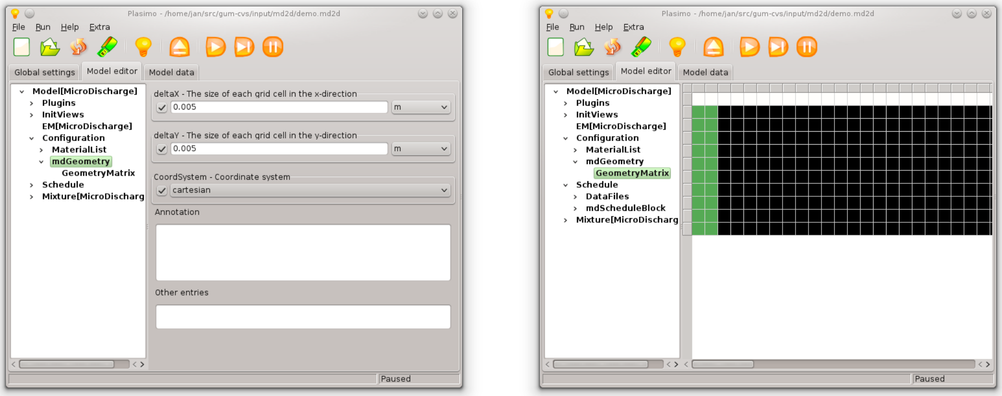
- Geometry - a Cylindrical coordinate system.
  To change the Cartesian system into a Cylindrical coordinate system is relatively easy. First, in the `Model editor` tab you must expand the `Configuration` node in the tree by clicking the "+" symbol. Then select `mdGeometry`. On the right-hand side the geometry parameters will be shown, see Fig. 4.1(a). From the drop-down menu `Coordinate system` select `cylindrical` instead of `cartesian`.

  Note: When a cylindrical geometry is used axis of symmetry is located at the bottom of the geometry matrix. In Cartesian coordinates the bottom row in the matrix represents a dielectric. In cylindrical coordinates this row must be removed. To change this, open the subsection `GeometryMatrix`. Please click on the lowest gray cell to the left of the matrix. This will active the last row. Next, right-click somewhere in the matrix and choose 'Cut section'. The resulting grid looks like that in figure 4.1(b).

- Distance between the electrodes.
  To change the distance between the electrodes, or the radius of the discharge, you must only change `deltaX` or `deltaY` (Fig. 4.1(a)).

  If you decrease the distance between the electrodes you should observe a decrease of the positive column length.

- Pressure.
  To change the pressure you must first select the `ModelEditor` tab, and then select `GasList`, which is located under the `Mixture` node.



(a) Geometry parameters (b) Geometry Matrix

Figure 4.1: Accessing the geometry parameters and matrix.

An increase of pressure causes all the negative zones (cathode dark space, negative glow, and Faraday dark space) to be compressed towards the cathode. A decrease of pressure naturally causes the reverse effect: the negative zones expand and their boundaries become more diffuse. The positive column is driven into the anode and disappears altogether when the pressure is sufficiently low.

- Potential and current.
  To define the potential as a function of time select `mdScheduleBlock` located in the `Schedule` node in the `Model Editor` tab. In the `Schedule` section time has been divided into ranges, so called "ScheduleBlock"'s, and for each of those one needs to specify the potential of each of the electrode materials. At the moment the order in which the different `Potentials` appear in the `ScheduleBlock` must match the order of the different electrode materials defined in the `MaterialList`!

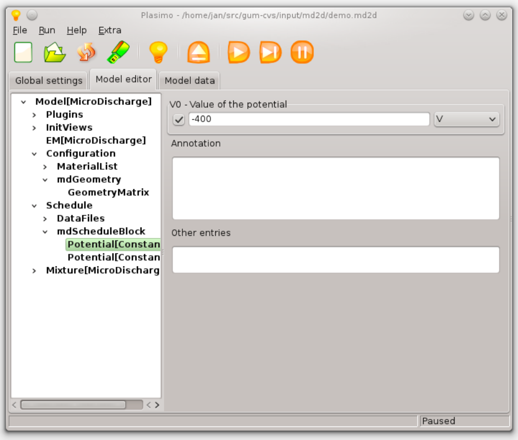


Figure 4.2: Setting the potential of the first electrode.

If the potential is raised the length of the cathode dark space decreases, whereas the length of the negative glow increases slightly. In reality there is also an increase in the current

through the discharge and consequently a general increase in the brightness of the luminous parts, since more energy is now being dissipated in the gas and some of this appears as additional radiation.

- Material of the cathode.
  The cathode has a marked effect upon the voltage necessary to maintain the discharge. Lower voltages are needed when the cathode is a good emitter of electrons under bombardment by positive ions or photons. You can play with this by changing the secondary emission coefficients. The secondary emission coefficients are specified for each type of species, and are located in the `Model editor` tab under `Mixture / mdSpeciesList / mdParticle`

4.3 Drift-diffusion sputtering model

Example input file: `hcd-demo.md2d`

Location: `input/md2d/`

The input file contains a model of a hollow cathode discharge (HCD) used for laser application. Description of the complete model you can find in [2, 3, 4], while here is a simplified version for demonstration purposes.

The geometry is a cylindrical hollow cathode with 4 mm inner diameter and 50 mm length; two anode rings are placed at both sides of the cathode, separated from each other with quartz rings. The geometry is symmetric around the center of the cathode (both axially and azimuthally).

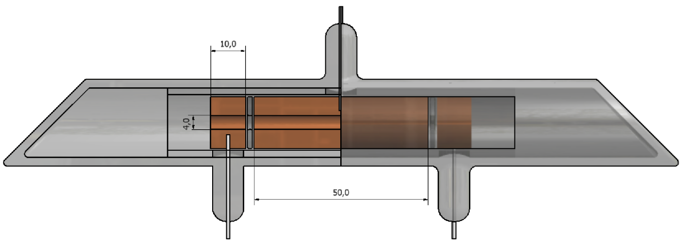


Figure 4.3: The hollow cathode geometry.

Hence, the simulated discharge region is half of the cathode length, the anode region and the dielectric in between. A fixed voltage of -300 V is applied.

The operating gas is He at constant pressure of 2.3 kPa. The considered species are He^+, Cu and Cu^+. The buffer gas atoms are ionized by electron impact ionization. The metal atoms in the discharge are produced only by sputtering due to ion bombardment of the cathode surface. The metal atoms are ionized in the plasma and they also can produce sputtering (self-sputtering). The ionization mechanism of Cu includes electron impact ionization and charge transfer with buffer gas ions.

Sputtering module

The sputtering process can be taken into account in the model and it can be applied at different materials if the geometry consists many. This can be done:

- In section `Mixture` with the right mouse bottom add section `mdWallProcessList`
- In `mdWallProcessList` you must add the process `mdWallProcess` of type sputtering (the only available so far).

- Select `mdWallProcess`; on the right side window you can fill the species bombarding the wall (the projectile) and the sputtered atoms (recoils). You have to specify also the material index where the process should be applied. The indices are specified in the input file in section `Configuration/Material list`. In this example you see the cathode with index 1, the anode region with index 2, the dielectric with index 3 and index 0 is reserved always for the discharge region. The sputtering process is applied on materials with index 1 and 2.

- The sputtering yield for every projectile species is specified by selecting `mdSputteringYield`. The subsection `mdSputteringYiled` appears automatically after adding `mdWallProcess` of type sputtering. The `mdSputteringYield` is of type `Constant`.

4.4 Monte Carlo model

Example input file: `mc-demo.in`

Location: `input/mc/`

This model is simple kinetic drift model. We have Ar gas (density swarm) and we inject 10000 particles (electrons) into the simulation. The particles are injected as point sources at coordinates (0,0,0). An uniform electric field is applied in z-direction.

The main branches of the tree structure in the `Model Editor` tab are:

- `Environment`: this node contains the setup of the vessel, specifies the particles that are injected into the vessel, and defines the swarm list, i.e. the environmental particles the injected particles will collide with (the background gas). In this example vessel of type `Infinity` is specified. This practically means that there is no vessel. The particles are moving under the influence of a constant electric field. The electric field is applied in z-direction (pointing to the screen). There are two more “Vessel” sections which are disabled. Vessel of type `plate` places a wall (a plate) at some specified distance (in z-direction) from the particle injection source. You can activate this section by pressing right mouse button over the disabled section.

- `Statistics`: obviously the necessary statistics are collected in this node, such as particle positions and velocities, and EDF’s of the particle swarm that is followed. Moreover, the `Statistics` class can collect statistics of wall collisions (if there is any wall), such as position and velocity at the moment when the particle hits the wall, as well as EDF and deposition profile.

- `mcSpeciesList`: This section contains a list of all species that are involved in the simulation, hence the model should know about.

- `FlightControl`: this node controls the “flight” of the particles, that must be followed, the chemical reactions that they are involved in and the time settings of the “flight”. Therefore, the `FlightControl` section contains `SwarmList`, `mcProcessList` and `TimeSettings` subsections.

After installing the model a tab `Model data` appears that contains all the output data. The output data is grouped into three main sections, as you can see on the left-hand side:

- `General`: containing the status of the calculation and the model-specific general data.

- `Collision frequency`: the collision frequencies for all collision processes.

- `Species`: positions, velocities and EDF for the followed particles. If specified in the input, the wall EDF or deposition profile of the followed particles will appear.

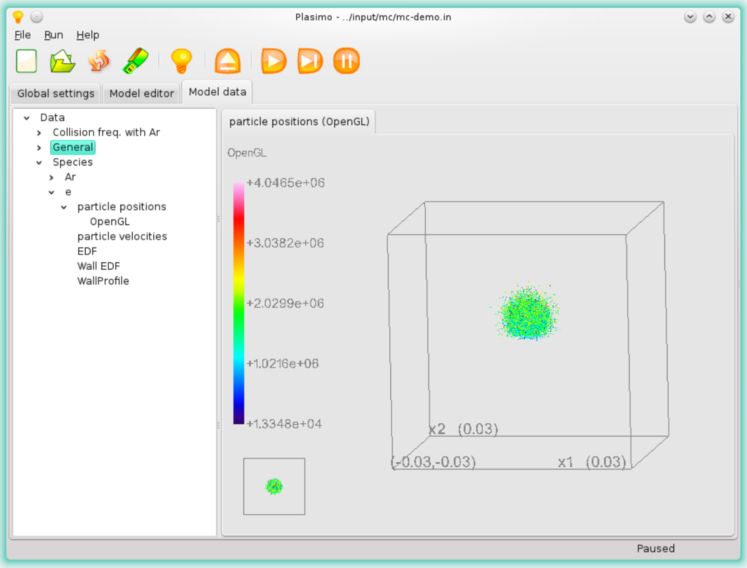


Figure 4.4: Monte Carlo model.

Few steps after starting the model you should see already the multiplication of the particles due to ionization processes. Recall that the electric field is applied in direction pointing to the screen. So that the electrons will move in opposite direction, away from the screen. To see clearly the moving of the particle cloud rotate the plot as shown in figure 4.4.

4.5 Argon cascaded arc model

Example input file: `ArArc8_KG.gum`

Location: `input/demosuite/`

This model deals with a plasma created in an arc by means of a DC current. The plasma source has the shape of a cigarette; it is a cylinder with a radius of about 2 mm and a length of about 60 mm. The current creating this plasma is about 50 A. We write "about" because you can play with these plasma control parameters and look what effect they have on the plasma properties, like the electron density and temperature (`ne`, `Te`), the gas temperature (`Tg`), and the flow field (velocities).
After loading the model, it can be explored by navigating through the leaves in the `Model editor`. The main sections that are present are:

- `MainIterConfig`: This section contains some parameters to control the iterations.
- `PlasmaRegion`: The input data is organized in the following subsections:
  - `Grid`: contains specifications of the geometry, coordinates and involved materials.
  - `TemperatureHP`: This section describes the heavy particle temperature. It contains a separate boundary condition (`BndConfig`) for every wall, and control of the iterations is handled in the `IterConfig` subsections.
  - `TemperatureEL`: Similar to `TemperatureHP` only now for the electron temperature.
  - `Mixture`: This describes the composition of the plasma, i.e. the species that are taken into account in the simulation and the relevant reactions that they are involved in.

– `Flow`: This section configures the flow field calculations.

- `EM`: The EM calculations. In this case it is an uniform current of 50 A .

Results and discussion

After installing the model in the `Model data` tab, you can see the run time plots and monitor the variables. All the calculated variables are organized in sections. The plots can be open from the tree structure from the left panel.

- `General`: gives the status of the calculation and the convergence log graph. During run time the graph gives the residue as a function of the iteration number;

- `plasma`: the left panel shows a tree structure containing a host of variables that are calculated by the model, organized in several groups: `Field(real-valued)`, `Field(vector)`, `Miscellaneous`, `Phi variable`, and `text`. The nodes under `Phi variable` are the primary quantities of interest.

The characteristics of this plasma source are the same as that of a positive column (PC). For this PC a global model can be made, which will have the general outcome:

- The electron temperature is (almost) independent of the power (density), but is dictated by the electron loss frequency. The latter is often "diffusive", meaning that this frequency more or less equals D_a/R^2, where D_a is the ambipolar diffusion coefficient, and R the plasma radius.

- The electron density (m^{-3}) is determined by the power density (Wm^{-3}).

- The heavy particle temperature is among others ruled by the electron density and the pressure (the plasma size...). So increasing n_e leads to higher T_h values.

4.6 Transonic jet model

Example input file: `transonicjet.gum`

Location: `input/demosuite/`

A model of a transonic jet with multiple shock waves is treated in this section. Description of the complete model and detailed information you can find in [5].

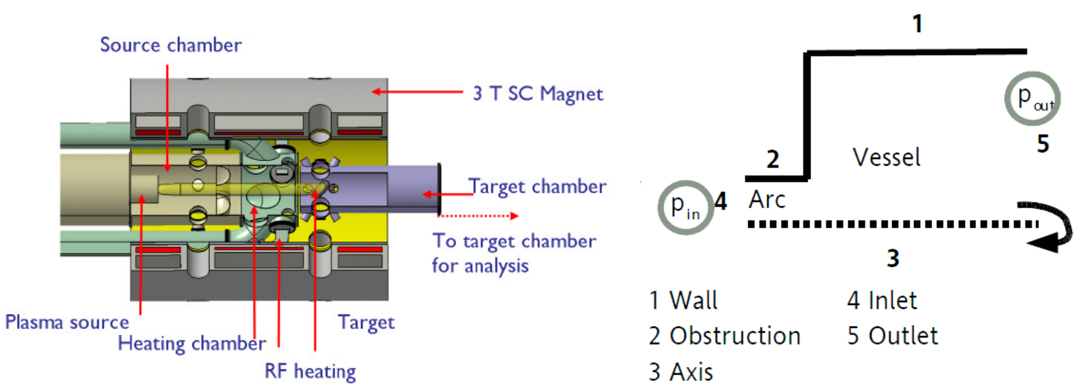


Figure 4.5: Geometry

Algorithm

- SIMPLE + Karki corrections
- Incompressible: $\Delta p \to \Delta v$
- Compressible: $\Delta p \to \Delta v / \Delta \rho$
- Karki corrections take into account influence p on ρ

Different flow regimes

The inlet and outlet pressure can be changed in the `Model editor` tab:
`Model/PlasmaRegion/Flow/Pressure/Function` (initialization)
`Model/PlasmaRegion/Flow/Pressure/BndConfig/BndCond` (boundary conditions)

The results can be seen in the Model data tab:
`Data/DefaultCylindricalGrid/Plasma/Barycentric`
`Data/DefaultCylindricalGrid/Plasma/Field(real-valued)/MachNumber`

- Set the Karki correction (`Model/PlasmaRegion/Flow/Pressure`) to 0.0: does the model still converge (`Data/General/Convergence log`)?
- What happens when you lower the inlet pressure; can you obtain a subsonic flow?
- Do subsonic flows converge without Karki corrections?

4.7 EM models

4.7.1 EM model - 1

Example input file: `circular_bessel.gum`

Location: `input/demosuite/`

The model represents transversal magnetic modes in a circular waveguide. The geometrical configuration is shown in figure 4.6.

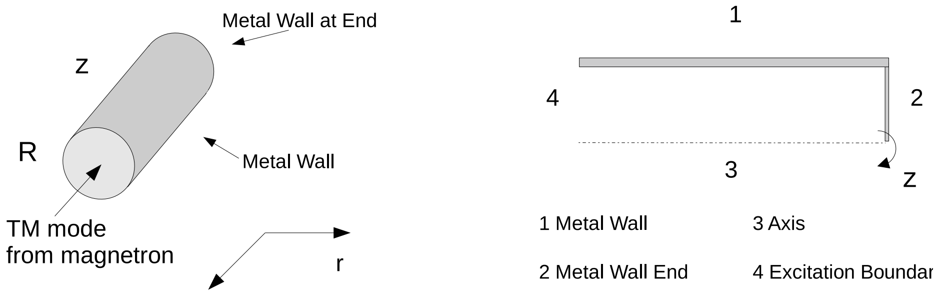


Figure 4.6: Geometry

Algorithm

- Finite Differences in Frequency Domain
- Full Vector TM (Transversal Magnetic): E_r ,E_z, $\tilde{H}_\phi$

Standing waves and Evanescent Solutions

Install the program, and run it. Plot the normalized magnetic field $\tilde{H}_\phi$
`Data/Cyl/Fields/em_region/Field (complex)/H_ud`

The excitation boundary creates a wave in z^+-direction, that interacts with the wave reflecting back from the metal wall at the end (2). Thus, standing waves are observed.

Below a critical radius, R_c, there are only evanescent (decaying) solutions instead standing waves. Decrease the radius R, until only evanescent solutions appear in the circular waveguide.

4.7.2 EM model - 2

Example input file: `radial_to_coaxial_wg_show.gum`

Location: `input/demosuite/`

The model represents radial-coaxial waveguide coupling. The geometrical configuration is shown on figure 4.7.

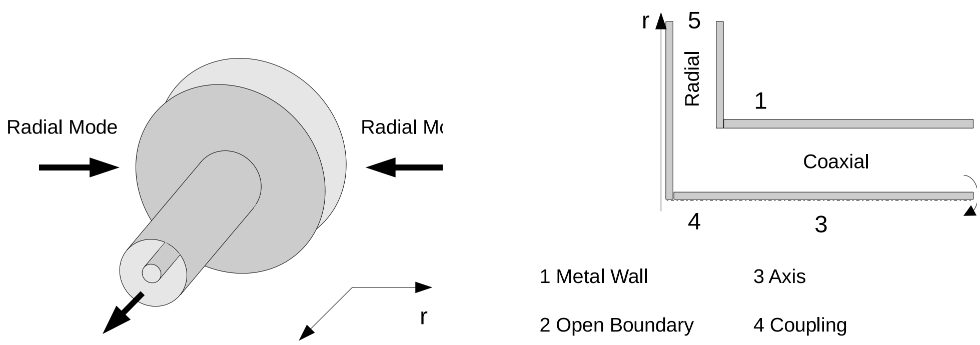


Figure 4.7: Geometry

Algorithm

- Finite Differences in Frequency Domain
- Full Vector TM (Transversal Magnetic): E_r ,E_z, $\tilde{H}_\phi$

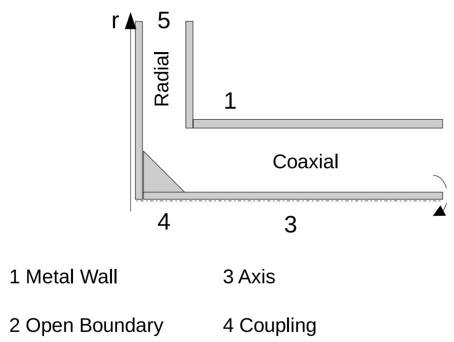
Conversion from Radial to Coaxial solution

Install the program, and run it. Plot the normalized magnetic field $\tilde{H}_\phi$
`Data/cyl/em_region/Field(complex)/H_ud`

The excitation boundary creates a wave in r^--direction. When it reaches the coupling, the wave is converted into a wave propagating in z^+-direction. An open boundary condition (2) prevents

most of the reflection, and almost no standing wave pattern is created in the coaxial waveguide. However, in the radial parallel plate waveguide, there is a standing pattern due to the reflection with the metal wire at the axis.

- Go to the Model editor and select in the hierarchical tree `Model/Grid/Geometry/CellMatrix`. You can now modify the geometry and draw a triangle (pyramidal cylinder in 3D), such as the one shown in the figure below.



  Right click and `Select Material`. Choose the `Metal`. Change the materials cells to metal by left clicking on them. Note that you can also enable the other section `CellMatrix`, where the triangle is already included.
  What happens to the standing waves in the radial waveguide?

- Change the boundary (2) from the open boundary condition (plane wave tangential E fields) to the homogeneous Dirichlet condition (null tangential E fields). You should notice a standing wave appearing in the coaxial waveguide.

4.8 Global Model

Example input file: `zdm_Ar.gum`

Location: `input/demosuite/`

This simple argon model is inspired by a model by Ashida et al. [6]. It is a Global Model that calculates the densities of species and reaction rates of processes between these species, as a function of time in a cylindrical geometry. Additionally, the electron energy density balance is solved. The Global Model calculates volume averaged quantities. It is therefore a type of Zero-Dimensional Model (ZDM), though this does not mean that the goal is to model a point in space, but rather some volume as a whole where no transport inside that volume is taken into account.

The main parts of the input file are the following sections:

1. *Species*
   The model contains three argon species (ground, excited, ion) and the electron species. The definition of the species is rather straightforward, and is mostly characterized by defining their unique name (by which they can be used in reactions), energy, elemental composition, and charge.

2. *VolumeRelations*
   Reactions (called `Relation` in Plasimo) are defined by a `Format` for the reaction formula and

a rate coefficient. Rate coefficients (called `Rate` in Plasimo) can be defined in many ways, such as Arrhenius rate coefficients and look-up tables. In this particular input file only rate coefficients of the type *CustomRate* are used. This is the most flexible type, since it accepts any mathematical expression (see the documentation for the mathparser for more details). For increased flexibility certain functions are available to use species properties in the rate coefficients, namely *Temperature()*, *Density*, *Mass()*, *Charge()*, and *Weight()*. In addition, the Global Model plugin allows the user to define constants and functions that can subsequently be used in expressions in *CustomRate* type rate coefficients. This will be explained in more detail in the section *Declares*. The *ApplyDB* option indicates whether detailed balancing needs to be applied to automatically include the reverse process. This will not result in an extra reaction, but only in a modified rate.

The processes included in the volume relations are (de)excitation, ionization, and radiation, the rates of which are adopted from the original model.

3. *WallRelations*
Wall relations are internally identical to Volume Relations. They are only included as a service to the user to distinguish between these two classes of processes. In this particular input file a different way of including transport of ions is used, as compared to what is described earlier in this manual where only extra sources were used. Here, an additional "dummy" species Ar_w^+ is used representing argon ions at the wall. Since this ion wall species has identical properties as the bulk ion species, the disappearance of ions to the wall can be written as a reaction, which now describes the transformation of bulk ion species to wall ion species. Since the wall ion species is not included anywhere else in the model, it has no further influence on the model. This separate wall species allows the user to track the total number of particles reaching the wall as a function of time. The other elements in this recombination process (return of neutral argon and energy loss) still need to be taken care of by introducing extra source terms.

   The second wall relation is the deexcitation of excited argon at the wall, as described earlier in this model.

4. *ExtraSources*
To complete the model three additional sources must be included, namely energy losses due to elastic collisions, kinetic energy losses at the wall, and neutral argon coming back from wall recombination. Again, all three source terms have a rate coefficient described by a *CustomRate*.

   Together with the *VolumeRelations* and the *WallRelations* all processes have been taken into account, so the plugin can construct the complete balance equations.

5. *Declarations*
The rate coefficients for processes can become quite complex. An example of this is the rate coefficient for ion recombination at the wall

$$K_{\mathrm{recomb},Ar^+} = \frac{A_{\mathrm{eff}}}{V} u_B, \tag{4.1}$$

   which in the model by Ashida when completely written out is

$$K_{\mathrm{recomb},Ar^+} = \frac{1}{\pi R^2 L} \left[0.86 \left(3.0 + \frac{L}{2 n_{\mathrm{neutral}} \sigma_i} \right)^{-\frac{1}{2}} 2\pi R^2 + 0.80 \left(4.0 + \frac{R}{n_{\mathrm{neutral}} \sigma_i} \right)^{-\frac{1}{2}} 2\pi R L \right] \sqrt{\frac{k_B T_e}{M_{Ar^+}}},$$

   where R and L are the radius and length of the cylindrical vessel and σ_i is the ion neutral collision cross section. As an expression in the Global Model input file this would be something like

```
(0.86*(3.0+L/(2*(Density('Ar')+Density('Ar*'))*sigma_i))^(-0.5)*
2*`pi*R^2+0.80*(4.0+R/((Density('Ar')+Density('Ar*'))*sigma_i))^(-0.5)*
2*`pi*R*L)*sqrt(`kB*Temperature('e')/Mass('Ar+'))/(`pi*R^2*L)
```

which is not very practical and error-prone. To make this more readable and manageable the user can define constants and functions, which can be used in rate coefficients or in other constants and functions, and so on. The difference between a constant and a function is that a constant will not change during the course of the model, while a function can. Additionally, a function can have parameters, namely zero, one, two, or three, i.e., a nullary, unary, binary, or ternary function, respectively. In `zdm_Ar.gum` this feature is used to first define a number of constants representing the geometry of the vessel (radius, length, areas, and volume), followed by a number of functions representing quantities that might change during the course of the model, such as the diffusion length, and Bohm velocity. The Bohm velocity function also takes as a parameter a species name so it can be applied to different species. Note that nullary functions are functions, so when used they must be accompanied by a set of empty parentheses. Also note that in principle a function can be used in the definition of a constant, but since the result is a constant it will not change and be fixed at the value at which it is initially evaluated. Furthermore, any of the definitions in *Declares* can be used in any of the rate coefficients (as long as they are of the type *CustomRate*). The definitions and their composition in this example are completely optional. For instance, the length is a constant, but it might also depend on some other quantity and therefore change. Finally, there is one variable called *time* which represent the current time of the model in seconds. Though it will change during the course of the model it is written as a constant, i.e., without parentheses, but it acts as a nullary function.

6. *InputPowerDensity*
   The power density input into the system must be defined in a separate section. This must always be an expression that evaluates to W/m^3 or compatible unit. In `zdm_Ar.gum` a trick is used to include an input power from a look-up table. The mathparser allows for function to be defined which are evaluated as a look-up in a table in an external file. In this case the file `zdm_power_modulation.lut` contains the power as function of time. By using the built-in *time* variable as index into the table and dividing the result by the Volume (see the appropriate Declare), an input power density as function of time is defined. Its shape is whatever the table looks like, using linear interpolation.

7. *InitDensity*
   This section contains the initial density of all the species present in the system. Each defined species must be listed. The sections ZDParameter are used to give the initial value of the electron temperature, and constant fixed value of the heavy particles, which all share the same temperature.

8. *ZDSchedule*
   This section simply states the start and time of the model. Note that it is not compulsory for the model to start at time zero.

9. *Stepper*
   The stepper is the ode solver that is to be used. Depending on the solver some additional options are available, such as maximum time step, tolerance, etc.

10. *ZDOptions*
    Finally, some general options for the model can be set. By default, for each defined species a balance equation will be constructed and solved. By adding a species to the *ConstantDensity* section the source term for that species will be set to zero so it will remain at its initial value. Furthermore, quasi-neutrality can be imposed, so the electron density is not solved separately, but set equal to the total ion density. Since it is possible to define ion species

that are not meant to be part of the bulk plasma (see the earlier ion wall species), the user must define which ion species are to be used to calculate quasi-neutrality, so that

$$\sum_i (q_i n_i) - n_e = 0, \tag{4.2}$$

is valid, where the sum runs over all ions i, with each having a charge q_i.

Finally, a minimum value of the electron density can be imposed. If this is enabled the electron temperature will not fall below the heavy particle temperature.

Appendix A

System requirements

the system requirements will be summarized for both Windows and Unix-like systems.

A.0.1 Requirements for Microsoft Platforms

At present, native compilation of Plasimo on Microsoft Windows systems is not supported. Instead, Windows binaries are made available using cross-compilation on Linux, using the `mingw` cross-compiler suite, see `http://www.mingw.org/`. Please contact the Plasimo team at `info@plasimo.phys.tue.nl` if you want to obtain a binary version, suitable for Windows XP. No additional libraries are needed if you want to install Plasimo this way.

A.0.2 Requirements for GNU/Linux and Unix Platforms

Table A.1 lists the packages which are needed to compile Plasimo on GNU/Linux and Unix systems. The end of this section gives detailed instructions on how to obtain these packages for the three most popular Linux distributions: Ubuntu, OpenSuse, and Fedora.

On most GNU/Linux installations you can check the availability of programs with the `which` utility. As an example, `which gcc` will print the location of the `gcc` program that is found in the path:

```
# which gcc
/usr/bin/gcc
```

If nothing is printed, the `gcc` program is not available and needs to be installed. Let us now assume that your system uses the `rpm` system to manage packages (like on Novell/SuSE, Redhat or Fedora systems). The version of the package that a file belongs to can be obtained as:

```
# rpm -qvf /usr/bin/gcc
gcc-4.1.0-25
```

In this case, `gcc` is present, the version being `4.1.0-25`.

The remainder of this section explains the purpose of the various packages. You may wish to skip this information and go to section 2, the Quick Installation Guide instead.

In order to resolve some platform incompatibilities between the various flavours of Unix during the build process, use has been made of the so-called GNU build tools. These are the programs gmake, aclocal, autoheader, automake, autoconf and libtool. In addition, you need a C/C++ compiler. An overview of these packages is given in table A.1. The version requirements listed there are pessimistic: older versions of these products may work, but have not been tested with this version of the code. Note that gmake is also known as make on many GNU/Linux and Unix platforms.

wxWidgets[1] is a cross-platform toolkit for building graphical user-interfaces. On GNU/Linux and Unix only the GTK version has been tested, obviously this is our recommended version (see

[1] wxWidgets was formerly known as wxWindows.

Product	Version	Location
gcc	4.5.1	`www.gnu.org/software/gcc/gcc.html`
automake	1.11.1	`www.gnu.org/software/automake/automake.html`
autoconf	2.68	`www.gnu.org/software/autoconf/autoconf.html`
libtool	2.2.6b	`www.gnu.org/software/libtool/libtool.html`
gmake	3.82	`www.gnu.org/software/make/make.html`
boost	1.50	`www.boost.org`
wxWidgets	2.9.0	`www.wxwidgets.org`
latex	3.141592-1.40.11-2.2	`www.ctan.org`
latex2html	1.71	`www.latex2html.org/`
pdflatex	3.141592-1.40-11-2.2	`www.tug.org/applications/pdftex/`
doxygen	1.7.3	`www.doxygen.org/`
dot	2.26.3	`www.graphviz.org`
python	2.7	`www.python.org/`
convert	6.6.5-8	`www.imagemagick.org`

Table A.1: System requirements for building the software on GNU/Linux platforms. The products wxWidgets and below are weak requirements, the console versions of the Plasimo applications can be built without them. wxWidgets is required for graphical user-interface support (recommended), the remaining items are required for generating the documentation locally (see text).

the wxWidgets home page for details). Note that you need not only the libraries but also the development (header files) and that using wxWidgets may introduce additional software dependencies. We recommend to use the official packages of your favourite Linux distribution, if available.

The packages latex, latex2html, pdflatex, doxygen, dot, python and ImageMagick (more specifically, the program `convert` which is part of it) are weak requirements: these are needed to locally build the complete documentation. For released versions of the code the documentation can also be obtained from the Internet. See chapter "Generation of the Plasimo Documentation" of the programmer's guide and the Plasimo website `http://plasimo.phys.tue.nl/generated-docs/` for details.

Some work has been spent to add support for other compilers than gcc, such as the Intel C++ compiler for Linux, the SGI MIPSpro compiler for IRIX and the Compaq/DEC CXX compiler for DEC Unix/Tru64 and Linux. In some case some special steps must be taken in the build process, this will be discussed in section B.2.4.

Ubuntu

The following instructions have been tested for Ubuntu 15.04 (a.k.a. Vivid). The required packages are listed in table A.0.2. The same instructions also work for Debian 8 (a.k.a. Jessie). Ubuntu 14.04 LTS has also been tested with this list of packages, however, an additional repository is required for the wxWidgets packages. This can be achieved with the following commands:

```
sudo apt-add-repository 'deb http://repos.codelite.org/wx3.0.2/ubuntu/ trusty universe'
sudo apt-get update
sudo apt-get install libwxgtk3.0-dev libglu1-mesa-dev
```

Note that the package `libwxgtk-webview3.0-dev` is not required, but the package `libglu1-mesa-dev` is.

OpenSuse

The required packages for version 13.2 are listed in table A.0.2.

Note, that it might be necessary to run the command `texhash` as root for the `html.sty` style file from the `latex2html` package to be found.

cvs	libtool	automake
autoconf	build-essential	gcc
g++	gfortran	libboost-all-dev
texlive-latex-base	latex2html	doxygen
graphviz	python	imagemagick
transfig	liblapack-dev	gnuplot
libwxgtk3.0-dev	libwxgtk-webview3.0-dev	tex4ht

Table A.2: Packages required for Ubuntu 15.04 and Debian 8 (approximately 1.5 GB).

cvs	gcc	gcc-c++
gcc-fortran	boost-devel	superlu-devel
automake	autoconf	libtool
make	wxWidgets-3_0-devel	Mesa-devel
texlive-latex	texlive-appendix	latex2html
gnuplot	doxygen	graphviz
python	ImageMagick	transfig
lapack-devel	texlive-tex4ht	texlive-comment

Table A.3: Packages required for OpenSuse 13.2 (approximately 1 GB).

Fedora

The following instructions are identical for Fedora versions 13 through 20. The required packages are listed in table A.0.2.

cvs	gcc-c++	gcc-gfortran
boost-devel	lapack-devel	automake
autoconf	libtool	make
texlive-latex	texlive-a4wide*	texlive-epstopdf*
texlive-subfigure*	texlive-appendix*	latex2html
gnuplot	doxygen	graphviz
ImageMagick	python	gtk2-devel
mesa-libGLU-devel	transfig	wxGTK-devel

Table A.4: Packages required for Fedora versions 13 through 20 (approximately 1 GB). The packages denoted by * are only required for release 18 and up.

Appendix B

Installation & Configuration

This section provide an in-depth discussion of the download and installation processes.

B.1 Downloading Plasimo

B.1.1 CVS

In order to facilitate the code development process and the management of multiple versions of it the CVS[1] revision control system has been used. In order to obtain the sources you will need to install this program if you do not yet have it. This section will discuss how to configure cvs, and how to use it to obtain a version of choice of the sources.

The cvs program provides numerous additional features to retrieve old versions, inspect the development history of files, comparing different versions by date, release identifier et cetera. Check out the on-line CVS manual[2] for details.

Setting up CVS

Assuming you have read access rights, you can use cvs to obtain the sources from the *CVS repository* on the Plasimo server, **plasimo.phys.tue.nl**. Because the server has been configured to accept secure (encrypted) connections only, you also need *secure shell (ssh)*, which will normally be installed on Unix/Linux machines[3].

The first step is to instruct your cvs client to use ssh. This can be done by setting the `CVS_RSH` environment variable to `ssh`. How this should be done depends on the shell you are using. Bourne-shell (bash) users do `export CVS_RSH=ssh`, while the C-shell (csh/tcsh) expects `setenv CVS_RSH ssh`. Check the manual of your shell if you are in doubt. It is convenient to have these variables set automatically by your shell's initialisation script, otherwise you need to set them every time you open a shell to start using cvs.

Checking out the Files

Now that we are equipped to use cvs, we may do an initial checkout of the sources. Firstly, switch to the directory where you want to install the sources. Then issue the command

```
cvs -q -d :ext:name@plasimo:/usr/local/src/cvs checkout gum
```

It may be necessary that you provide the fully qualified name of the server instead of just **Plasimo**, namely **plasimo.phys.tue.nl**. Furthermore, replace `name` with your login name on **Plasimo**.

[1] see `http://www.cvshome.org/`

[2] see `http://cvsbook.red-bean.com/`

[3] When using Microsoft Windows a program such as TortoiseCVS (`www.tortoisecvs.org`) can be used, or even an environment such as Cygwin (`www.cygwin.com`).

Please note that all commands and options under Unix are case-sensitive. This will create the directory `gum` in the current working directory. The option `-q` instructs cvs to report non-trivial output only. If your connection to the cvs server is via a modem, you should specify `-z3` before `-q` in order to save time by compressing all data traffic.

You can specify a set of default cvs options by creating a file `.cvsrc` in your home directory. A typical `.cvsrc` looks as follows:

```
cvs -z3 -q
update -d -P
diff -u
```

The meaning of the last two lines, which configure the defaults for the `update` and `diff` subcommands will become clear in section B.1.1.

By default, the subcommand checkout of cvs will give you the HEAD version of the sources. As explained in section 2 this is the branch on which active development takes place, the code here is not guaranteed to function properly, or even compile! Unless you have developer ambitions yourself, you should therefore check out one of the stable releases, probably the latest.

For getting such a stable release of the Plasimo sources, replace the part `checkout gum` in the cvs command above by `checkout -r REVISION gum`. Here `REVISION` should be replaced by the symbolic name of the version you want to fetch. As an example, the symbolic name `plasimo_4_0_0` corresponds to version 4.0.0 of Plasimo. These tag names can be found on the Manual and Source Code Documentation page on the website `http://plasimo.phys.tue.nl/`.

In case you have a version of the code already and want to install a second one, rename the gum directory to `gum-2.8.1`, say, before fetching the second version by entering another cvs checkout command.

Updating and Switching to Another Version

Before updating to the latest version or switching to a different stable release, you may want to inspect the changes you have made locally. The reason is that these changes may conflict with those made by others and checked in. This can be achieved by changing to the gum source directory, and entering

```
cvs diff -u > mychanges.diff
```

This command writes the differences you have made locally to a file called `mychanges.diff`. Inspect this file to see if you made any important changes that you don't want to get lost. If this is the case, the best thing is to put the relevant files in a safe place, before continuing. In the file some lines may start with question marks. Unless you willingly added files locally you can ignore these lines. If you want to discard your changes, you now delete the locally modified files.

If you have the HEAD version of the Plasimo sources installed already, you can merge the latest version of the source and documentation files from the server with your local working copy by switching to the gum directory and typing

```
cvs update -d -P
```

The update-specific option `-d` indicates that newly created directories are checked out as well; `-P` means that files which are no longer in the repository are removed from your local gum directory[4]. In order to switch to a version other than HEAD you use the same command, but in addition you specify the version to check out using the `-r` option:

```
cvs update -d -P -r ReleaseID
```

Where ReleaseID is to be replaced with the symbolic name, like `PLASIMO_2_0`. Finally, if you are using a specific version of Plasimo, but want to switch to the HEAD version, use the option `-A` once with the update command:

[4]Because this is almost always what you want, you best add the line `update -d -P` to the file `.cvsrc` in your home directory. The same holds for the line `diff -u`.

```
cvs update -A
```

In following update commands the option `-A` can be omitted.

We end with an important note: after updating Plasimo, *always* check the `NEWS` file for changes that may affect you.

B.2 Building the GUM binaries

B.2.1 Compilation on Linux/Unix Platforms

Here it is assumed that your compiler and linker are supported by the GNU build toolchain (the programs autoheader, autoconf, automake and libtool). This is definitely the case if you have a GNU/Linux system and use the `gcc` compiler. If you have another platform or compiler some tweaking may be required. In that case please read the notes in the next section. Here's the complete build recipe:

1. Switch to the *top level source directory*, `gum`.

2. Execute `./makeconf.sh`. This shell script will execute various GNU build tools in the proper order. Some steps are repeated in subdirectories. This may take a while (in the order of a minute). You may get a warning telling you that you should add m4 macros to aclocal, which you can safely ignore. If the script fails, you may need to call it from a Bourne shell by invoking `/bin/sh` or `bash` before calling the script.

3. Create a *build directory* and switch to it. This is where you are going to compile the libraries and application. You can have multiple build directories, and such directories can be added and removed at any time. It is best to give them names which characterize the configuration they will contain, like `linux-debug`, `MipsPro-optimized` or `win32-cross`.

4. From the *build directory*, run `configure`. This script is located in the top-level source directory `gum`. Configure accepts various options, some of these are general, others specific to the gum source package. You can run configure with `--help` as single argument to get a list of available switches, the gum-specific are listed below. The default settings are shown as is, if you want to override one or more of these defaults, you need to explicitly pass the options between brackets to configure.

   - `(--with-wx[=wx-bindir]), --without-wx`:
     If you enable this option, the Plasimo applications with a graphical user interface will be built. This requires a recent version of the wxWidgets toolkit, a cross-platform GUI-library. If no argument is provided, configure will look for a file `wx-config` in the directories in the `PATH` environment variable, and use that one. You may specify an alternative version of wxWidgets by providing the name of the directory in which the corresponding `wx-config` resides, as in `--with-wx=/usr/local/wx-HEAD/bin`.
   - `(--with-opengl), --without-opengl`:
     Building with OpenGL support enables some Plasimo extensions, among which a variety of viewers for two- and three-dimensional data sets. Note that this option can only be enabled if you have OpenGL libraries and headers properly installed. It also implies usage of a version of wxWidgets with OpenGL extensions enabled. If you enable OpenGL, and the configure script finds the Fame library[5] on your system it is possible to create MPEG clips from OpenGL windows.
   - `--with-samples, (--without-samples)`:
     By default, all samples and tests will be built. If you disable this feature, you can still create the samples by switching to the sample directory of your choice and do this part of the build manually, as in `cd plspecies/samples/particle; gmake`

[5] see `http://fame.sourceforge.net/`

- `(--with-doc), --without-doc`:
  If you enable this option, the documentation is created. This is a time-consuming process, and requires some auxiliary programs (doxygen, graphviz). If you can afford it, read the Plasimo documentation online on our web site `plasimo.phys.tue.nl/plasimo-doc/` (login required).
- `(--enable-static), --enable-shared,`
  `--disable-static, (--disable-shared)`:
  The default setting is to build dynamic libraries only. Plasimo relies on shared library loading, so use this switch to change this behaviour only if you know what you are doing.
- `--enable-debug, (--disable-debug)`:
  By default debugging information will be generated. This is useful during code development, but significantly increases the size of the resulting libraries and applications. If you don't know how to operate a debugger, you can safely disable this option.
- `(--enable-optimize), --disable-optimize`:
  By default, the code is compiled without optimizations. Production versions of the code should be compiled with optimizations enabled: it makes a huge difference in execution speed. Note that on most platforms optimization prevents debugging, so you might want to have two build trees, one with optimization for fast runs and one with debugging information but without optimization. GNU's gcc compiler suite allows to enable debugging optimized code, but note that inlining reduces the visibility of symbols from within the debugger.
- `(--enable-opt-arch), --disable-opt-arch`:
  Using this flag enables optimizations for the specific processor present on the machine where the code is built. This can improve the performance of the resulting code by a few percent, but may prevent the binaries to work on machines with a different (older) CPU.
- `(--enable-colors), --disable-colors`:
  When `Log(X)` messages are output, these may can colored according to their loglevel, red for 3, yellow for 2, green for 1 and bright white for 0. The application outputs escape sequences to set the colors, which work on most terminal emulations, basically everything that's compatible with a vt100 in this respect. Use the option to get more easily readable output if your terminal emulation supports it (both X11 xterm and Linux console do).
- `(--enable-smp), --disable-smp`:
  The TBCI numerics library used supports distributing large vector or matrix operations amongst several local CPUs by using POSIX threads. This speeds up operations for *large* objects on multiprocessor (SMP) machines. (For small data sizes, TBCI just does not make use of multithreading.) Use this flag to build with support for this; it can be controlled a run-time via the `Threads` or `MaxThread` configuration options.

Running `configure` takes some time, up to a couple of minutes. If configure fails, it will normally due to an ill-formed list of options, and tell you how to correct. Typical cases are: missing required libraries, or attempting to enable OpenGL without wxWidgets. You may check `config.log` if you need more details.

5. The next step is to run `make`. You need GNU make, note that on some systems this is available under the name `gmake` instead. This will create the required libraries and the Plasimo main applications. Depending on the options which were passed to `configure`, the wxWidgets-based graphical front-end, the OpenGL extensions, the documentation and the executables in the samples subdirectories will be compiled as well. Note that a fresh build takes a considerable amount of time, perhaps an hour or more.

If you have updated to a different version of Plasimo (see B.1), you are safe if you repeat all the steps above. You may save some time, however:

- If you know that none of the `configure.in` files have changed, you can skip the `./makeconf.sh` step. In case of doubt, always run `./makeconf.sh` or `gmake -f Makefile.dist` in the gum source directory.
- If you want to recompile with the same configuration options as before, you don't need to create a new build directory, but you can use one of those that you created before. Change to your build directory (whether it's new or not).
- If you are sure that none of the `Makefile.am` files have changed, you can as well skip the `../configure` step. As above: In case of doubt rerun the `../configure` command.
- If you run `../configure` again in an already existing build directory, you probably want to use the same configuration as before. To this end `configure` has been made to emit the script `reconfigure` in the build directory. This will invoke `configure` with the same arguments as the last time it was called.
- Recompilation with `gmake` should require significantly less time than the original make, unless lots of files have changed or some of the headers that are included by almost any file have changed.

If you encounter weird errors after an update and a partial rebuild, switch to the build directory and enter:

```
make clean
make
```

The reason for the problems may have been an error in the dependency calculation. As a result some files may not have been rebuilt erroneously, these two commands guarantee that all files will be recompiled.

B.2.2 Support for the Eigen Matrix/Vector Library

Plasimo can be compiled with support for the Eigen library. That provides an alternative for the TBCI matrix/vector classes. In order to enable Eigen-specific components, you need to download a recent version of Eigen, install that and pass additional compilation options to Plasimo. For that you may first need to install the package 'mercurial' (on OpenSuse), which provides the command `hg`, and the package `cmake`. On other distributions, the package names may be different. Then proceed as follows:

```
cd src
hg clone https://bitbucket.org/eigen/eigen/
cd eigen
mkdir linux-opt
cd linux-opt
ccmake ..
```

In the cmake window, change the PREFIX variable to something like `/home/jan/local/eigen` (of course, replace this path with something that matches your own set-up). Then press c (configure) until the option g (generate) appears. Press 'g' to generate the makefiles and exit. Then you can build and install the code:

```
make
make install
```

Now you are ready to confire Plasimo with eigen support. We recommend that you use a new build directory for this Plasimo flavour, as in the following example:

```
cd /home/jan/src/gum
mkdir linux-opt-eigen
cd linux-opt-eigen
../configure --with-eigen=/home/jan/local/eigen/include/eigen3
```

(You may wish to pass your other favourite options to the configure script as well.) If you also want to use eigen *instead of TBCI* for the linear systems code, you configure as follows:

```
../configure --with-eigen=/home/jan/local/eigen/include/eigen3 --with-eigen_for_linsys
```

Next, build the sources as usual with the `make` command.

B.2.3 Compilation on OSX Platforms

Plasimo —including its graphical user-interface— can be compiled on OSX platforms using the native `clang` compiler. A brief summary of the steps follows, which was tested on an OSX-10.10 (Yosemite) box. The `clang` compiler is available through Apple's XCode Developer Tools, which can be downloaded in the app store. After installation make sure you agree to the XCode license agreement (sudo is a command that allows the execution of a command as superuser): `sudo xcodebuild -license`

Building Plasimo also requires a fortran compiler. There are several possibilities, but the one we tested was the gfortran compiler package from https://gcc.gnu.org/wiki/GFortranBinaries#MacOS (version 5.1). Your system's security settings might prevent installation of this package, which can be amended under "Security & Privacy" in your "System preferences" by temporarily setting "Allow apps downloaded from:" to "Anywhere".

Apart from the two compilers, a number of additional tools a required, which can be installed using the MacPorts project; similar projects such as Fink and homebrew have not been tested.

The easiest way to install MacPorts is by downloading the appropriate dmg from the website (http://www.macports.org), opening it and running the installer package (.pkg). This sets up a minimal install (in /opt/local/) and sets required environment variables. The `port` command can then be used to install MacPorts packages. Make sure XCode is installed and you have agreed to the license agreement before installing MacPorts.

Interaction with MacPorts and compilation of Plasimo is done from the command line, using a terminal. OSX is equipped with a terminal emulator which can be found in Applications/Utilities.

MacPorts are continually updated (similar to Linux distribution repositories). Bringing your MacPorts install up to date can be achieved by issuing the command: `sudo port -v selfupdate` followed by: `sudo port upgrade outdated`.

To install a MacPorts package the following command must be used: `sudo port install <packagename>` Table B.2.3 lists the required packages.

cvs	automake	autoconf
libtool	boost	f2c
wxWidgets-3.0	graphviz-devel	
doxygen	texlive-latex	texlive-latex-extra
texlive-fontutils	texlive-htmlxml	latex2html
gnuplot	transfig	ImageMagick

Table B.1: MacPorts packages required for OSX (approximately 4 GB).

The `libtoolize` binary is installed as `glibtoolize`, which can be fixed by adding a symbolic link: `sudo ln -s /opt/local/bin/glibtoolize /opt/local/bin/libtoolize`

The code below assume a user named `jan` who sets up the sources of the required packages in the subdirectory `src` of his home direcory `/Users/jan`, say. Please replace the directory name `/Users/jan/src` in the instructions below with another directory — every directory will do, as long as you have write access to it.

Download, compile and install Plasimo

```
cd /Users/jan/src
cvs -d :ext:USERNAME@plasimo.phys.tue.nl:/usr/local/src/cvs checkout gum
cd gum
cvs update -dP
./makeconf.sh
mkdir mac-opt
cd mac-opt
../configure --prefix=/Users/jan/plasimo-opt \
  --with-wx=/opt/local/Library/Frameworks/wxWidgets.framework/Versions/wxWidgets/3.0/bin \
  CPPFLAGS=-I/opt/local/include
make
make install
```

If evrything works out Plasimo is now compiled and installed in the directory `/Users/jan/plasimo-opt`. Let's now test it:

```
cd /Users/jan/plasimo-opt
cd bin
source set-plasimo
./wxplasimo
```

Instead of the last command, you could also have typed `./plasimo`, followed by the name of an input file, to start the console version. The input files were installed in `/Users/jan/plasimo-opt/input`.

B.2.4 Miscellaneous Compilers on Linux/Unix

Other compilers than gcc have been used to compile Plasimo on Unix-like systems, like DEC/-Compaq CXX, Intel C++ and SGI MipsPro. The following remarks apply when using compilers other than gcc in order to build Plasimo.

When calling `configure`, you can specify the C-Compiler by setting the environment variable `CC` and the C++-Compiler by using the environment variable `CXX`. The linker is specified by the variable `LD`. If the compiler needs a certain flag to operate properly, you can include it in the variable as well.

During the configure process, test programs will be run which check for certain compiler features and bugs; unfortunately it's difficult to preview and/or work around all of these. Some bugs or features may be influenced by additional compiler flags. Some of them are already known to configure and automatically added to the compiler flags. Some may not and need to be added at compile time. Use the make options `EXTRA_CFLAGS`, `EXTRA_CXXFLAGS`, and `EXTRA_LDFLAGS` to add extra flags for the C compiler, the C++ compiler and the linker respectively. As a typical example, for some GNU/Linux systems you need `EXTRA_CXXFLAGS=-D_BSD_SOURCE` if you compile with wxWidgets enabled, because wxWidgets uses string functions that are only defined on BSD like systems (and the GNU system defines them only if `_BSD_SOURCE` is set. As a last resort, you may need to fix the `Makefile`s if your compiler is too different from the GNU compiler.

Here's a example for compilation on a Silicon machine running IRIX and using the MIPSpro compiler (version 7.3.1.1m). The text is based on an e-mail by Bart Hartgers, who figured this all out:

1. Go to your build directory.

2. Create the standard C++ headers that MIPSpro is missing by typing

```
mkdir cmissing
cd cmissing
python ../../util/generate_cpp_c_headers.py
cd ..
```

Obviously, you need the python[6] interpreter installed in order to have this work. Otherwise ask us for a tarball with the missing headers for MIPSpro.

3. Call configure with specifying Linker, C- and C++-Compiler.

```
LD="CC" CC="cc" CXX="CC␣-I‘pwd‘/cmissing" ../configure \
   --enable-isoc99 --enable-optimize --disable-debug
```

Note: This syntax will only work on a Bourne shell.

4. Call a script to fix the Makefiles for MIPSpro.

```
../util/fix-Makefile-mipspro
```

5. Compile specifying some extra flags:

```
EXTRA_LDFLAGS='-elf␣-shared␣-lC' \
EXTRA_CXXFLAGS='-g␣-O2␣-exceptions␣-ptused␣-FE:template_in_elf_section' \
EXTRA_CFLAGS='-O2' gmake
```

You can omit the `-O2` in case you don't want your compiler to optimize the generated code. In case your compilation aborts with an error, you may still want to compile the rest by using `gmake -k` instead of `gmake`.

6. The MIPSpro compiler needs different flags for the executables in the end. So, go to the `app/` directory and do

```
gmake clean
EXTRA_LDFLAGS='-call_shared␣-lC' \
EXTRA_CXXFLAGS='-g␣-O2␣-exceptions␣-ptused␣-FE:template_in_elf_section'
EXTRA_CFLAGS='-O2' gmake
```

7. The compiler seems to make a mess of functions to call with plComplex arguments. I can't really figure out what is going on exactly.

Bart Hartgers concluded his email with the following words: "Yes, this is awful. It was much more so to figure this out. Expect things to get better in the future as some of this knowledge may be built into the autoconf/make machinery."

B.3 Cross-Compilation on GNU/Linux for MSW Platforms

At present, native compilation of Plasimo on Microsoft Windows systems is not supported. Instead, Windows binaries are made available using cross-compilation on Linux, using the `mxe` cross-compiler suite, see `http://mxe.cc/`.

Cross-compilation involves three steps:

- Downloading, patching[7] and compiling the cross compiler itself;
- Using the cross compiler to create the MSW-libraries that Plasimo must be linked against (such as the wxWidgets toolkit);
- Cross-compile Plasimo itself.

The first two steps are facilitated by a makefile that we have provided in the sub-direcroty `util/` of the Plasimo source tree. Please take a look at `util/Makefile.cross`; the prerequisites for building a cross compiler are listed in the beginning of that file. After installing the missing ones on your system, you can set up the cross compilation environment with the following commands:

[6] see `http://www.python.org/`

[7] In order for shared libraries to work correctly, the option `--enable-fully-dynamic-string` MUST be passed to the compiler. Ths is achieved by patching the file `mxe/src/gcc.nk` that is part of the MXE distribution.

```
cd /home/jan/src/gum/util
make -f Makefile.cross build_mxe
```

This will set up the required sources in `$(HOME)/local/src/mxe` and install cross compilers for 32 and 64 bit Windows in the directories `$(HOME)/local/mxe/win32` and `$(HOME)/local/mxe/win64`, respectively. If you prefer different paths, edit the relevant lines in the top of `Makefile.rules`.

The next step is to cross-compile `wxWidgets` with the freshly created cross compilation tools. This can be achieved with the command

```
make -f Makefile.cross build_wx
```

This will set up the required sources in `$(HOME)/local/src/wxWidgets-head` and install 32 and 64 bit Windows versions of the `wxWidget` headers and libraries in `$(HOME)/local/mxe/win32/wx-head` and `$(HOME)/local/mxe/win64/wx-head`. Also debug-versions of these libraries will be created in subdirectories `wx-head-dbg`.

Now it is time to cross-compile Plasimo. For the 64-bit Windows version, do (change the prefix to something else than `/home/jan/...`:

```
cd gum
mkdir mxe-win64
export PATH=/home/jan/local/mxe/win64/bin:$PATH
cd mxe-win64
../configure --host=x86_64-w64-mingw32.shared --without-samples \
    --prefix=/home/jan/gum-mxe-win64 --disable-inst-libs \
    host_alias=x86_64-w64-mingw32.shared \
    --with-wx=/home/jan/local/mxe/win64/wx-head/bin
make winzip
```

In order to compile the 32-bit version:

```
cd gum
mkdir mxe-win32
export PATH=/home/jan/local/mxe/win32/bin:$PATH
cd mxe-win32
../configure --host=i686-pc-mingw32.shared --without-samples \
    --prefix=/home/jan/gum-mxe-win32 --disable-inst-libs \
    host_alias=i686-pc-mingw32.shared \
    --with-wx=/home/jan/local/mxe/win32/wx-head/bin
make winzip
```

The result will be a ZIP file that contains Plasimo and the required support libraries. If you have wine installed, you can test this version of Plasimo as follows. Starting from the build directory, type (replace ¡version¿ with the version identifier):

```
mkdir tmp
cd tmp
unzip ../plasimo-<version>.zip
cd plasimo-XXX
wineconsole cmd
```

In the command window that pops up, type `Plasimo` (GUI version) or `CPlasimo`, followed by an input file (console version).

Bibliography

[1] Jan van Dijk, Kim Peerenboom, Manuel Jimenez, Diana Mihailova, and Joost van der Mullen. The plasma modelling toolkit plasimo. *Journal of Physics D: Applied Physics*, 42(19), 194012 (14pp), 2009.

[2] D. Mihailova, M. Grozeva, G.J.M. Hagelaar, J. van Dijk, W.J.M. Brok, and J.J.A.M. van der Mullen. *J. Phys. D: Appl. Phys.*, 41, 245202, 2008.

[3] D. Mihailova, J. van Dijk, M. Grozeva, G.J.M. Hagelaar, and J.J.A.M. van der Mullen. *J. Phys. D: Appl. Phys.*, 43, 145203, 2010.

[4] D.B. Mihailova. PhD thesis, Eindhoven University of Technology, The Netherlands, 2010.

[5] K S C Peerenboom, W J Goedheer, J van Dijk, and J J A M van der Mullen. Integral simulation of the creation and expansion of a transonic argon plasma. *Plasma Sources Science and Technology*, 19(2), 025009 (9pp), 2010.

[6] S. Ashida, C. Lee, and M.A. Lieberman. Spatially averaged (global) model of time modulated high density argon plasmas. *American Vacuum Society*, 13(5), 2498, 1995.